METHOD AND SYSTEM FOR UNAMBIGUOUS ADDRESSABILITY IN A DISTRIBUTED APPLICATION FRAMEWORK IN WHICH DUPLICATE NETWORK ADDRESSES EXIST ACROSS MULTIPLE CUSTOMER NETWORKS

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to an improved data processing system and, in particular, to a method and system for multiple computer or process coordinating. Still more particularly, the present invention provides a method and system for network management.

2. Description of Related Art

Technology expenditures have become a significant portion of operating costs for most enterprises, and businesses are constantly seeking ways to reduce information technology (IT) costs. This has given rise to an increasing number of outsourcing service providers, each promising, often contractually, to deliver reliable service while offloading the costly burdens of staffing, procuring, and maintaining an IT organization. While most service providers started as network pipe providers, they are moving into server outsourcing, application hosting, and desktop management. For those enterprises that do not outsource, they are demanding more accountability from their IT organizations as well as demanding that IT is integrated into their business goals. In both cases, "service level agreements" have been employed to contractually guarantee service delivery between an IT organization and its customers. As a result, IT teams now require management solutions that

focus on and support "business processes" and "service delivery" rather than just disk space monitoring and network pings.

IT solutions now require end-to-end management that includes network connectivity, server maintenance, and application management in order to succeed. The focus of IT organizations has turned to ensuring overall service delivery and not just the "towers" of network, server, desktop, and application. Management systems must fulfill two broad goals: a flexible approach that allows rapid deployment and configuration of new services for the customer; and an ability to support rapid delivery of the management tools themselves. A successful management solution fits into a heterogeneous environment, provides openness with which it can knit together management tools and other types of applications, and a consistent approach to managing all of the IT assets.

With all of these requirements, a successful management approach will also require attention to the needs of the staff within the IT organization to accomplish these goals: the ability of an IT team to deploy an appropriate set of management tasks to match the delegated responsibilities of the IT staff; the ability of an IT team to navigate the relationships and effects of all of their technology assets, including networks, middleware, and applications; the ability of an IT team to define their roles and responsibilities consistently and securely across the various management tasks; the ability of an IT team to define groups of customers and their services consistently across the various management tasks; and the ability of an IT team to address, partition, and reach consistently the managed

devices.

Many service providers have stated the need to be able to scale their capabilities to manage millions of devices. When one considers the number of customers in a home consumer network as well as pervasive devices, such as smart mobile phones, these numbers are quickly realized. Significant bottlenecks appear when typical IT solutions attempt to support more than several thousand devices.

Given such network spaces, a management system must be very resistant to failure so that service attributes, such as response time, uptime, and throughput, are delivered in accordance with guarantees in a service level agreement. In addition, a service provider may attempt to support as many customers as possible within a single network management system. The service provider's profit margins may materialize from the ability to bill the usage of a common network management system to multiple customers.

On the other hand, the service provider must be able to support contractual agreements on an individual basis. Service attributes, such as response time, uptime, and throughput, must be determinable for each customer. In order to do so, a network management system must provide a suite of network management tools that is able to perform device monitoring and discovery for each customer's network while integrating these abilities across a shared network backbone to gather the network management information into the service provider's distributed data processing system.

Hence, there is a direct relationship between the ability of a management system to provide network

monitoring and discovery functionality and the ability of a service provider using the management system to serve multiple customers using a single management system. Preferably, the management system can replicate services, detect faults within a service, restart services, and reassign work to a replicated service. By implementing a common set of interfaces across all of their services, each service developer gains the benefits of system robustness. A well-designed, component-oriented, highly distributed system can easily accept a variety of services on a common infrastructure with built-in fault-tolerance and levels of service.

Distributed data processing systems with thousands of nodes are known in the prior art. The nodes can be geographically dispersed, and the overall computing environment can be managed in a distributed manner. The managed environment can be logically separated into a series of loosely connected managed regions, each with its management server for managing local resources. The management servers coordinate activities across the enterprise and permit remote site management and operation. Local resources within one region can be exported for the use of other regions in a variety of manners.

However, prior art solutions for managing large, highly distributed data processing systems encounter significant problems while attempting to provide service to multiple customers. As noted above, a service provider's management system must be able to perform device monitoring and discovery for each customer's network while integrating these abilities across multiple customer networks. A customer generally wants remote

monitoring and management of its own network but also wants to be able to administer its own network in certain aspects as if the network is a dedicated, closed system, which generally means that a customer's network is shielded behind firewalls. Maintaining protection of individual networks presents a significant barrier to a service provider in accomplishing efficient network management.

Moreover, a service provider must address other problematic issues. If a customer is outsourcing certain functions to a service provider, the customer generally does not want to completely replace entire systems, so a service provider must have a solution that can be implemented on legacy systems and does not require the replacement of an entire IT infrastructure. If a service provider has multiple customers, and each customer has legacy systems, then the service provider is confronted with implementing a network management solution across a diverse, heterogeneous computing environment.

One particular problem that the service provider must confront is the fact that many customers may have software-based and/or hardware-based network address translators, or NATs. Each network address within a given domain serviced by a NAT can be assumed to be unique. However, across multiple NATs, each network address within the entire set of network addresses cannot be assumed to be unique. In fact, the potential for duplicate addresses over such a large, highly distributed network is quite high. Even if the service provider is managing only one customer within a particular network management environment, the same problem might also exist because a single customer may operate multiple NATs for

multiple networks. Given the fact that the service provider may be confronted with the conglomeration of multiple customer systems that have different types of NATs and operating systems, the solution needed by a service provider must be rather robust.

Therefore, it would be particularly advantageous to provide a method and system that provides a flexible network management framework in a highly distributed system with significant potential for duplicate addresses such that the network management framework can handle the intermingling of addresses from multiple customer networks. It would be particularly advantageous for the network management system to provide the ability to manage multiple customers within a single logical network.

SUMMARY OF THE INVENTION

A method, system, apparatus, and computer program product is presented for management of a distributed data processing system. A request for an action at a target device within the distributed data processing system is received; the request for an action at the target device uniquely identifies the target device using a system address for the target device, yet completion of the action depends upon a network address of the target device within the distributed data processing system. In response to a determination that a second device within the distributed data processing system has a network address that duplicates the network address of the target device, the duplicate network address is presented to a user along with other system address information for the target device and the second device. The user enters a virtual private network identifier (VPN ID), which is incorporated into the system address of the target device, and the execution of the requested action is then permitted to resume.

BRIEF DESCRIPTION OF THE DRAWINGS

The novel features believed characteristic of the invention are set forth in the appended claims. The invention itself, further objectives, and advantages thereof, will be best understood by reference to the following detailed description when read in conjunction with the accompanying drawings, wherein:

Figure 1A is a diagram depicting a known logical configuration of software and hardware resources;

Figure 1B is a block diagram depicting a known configuration of software and/or hardware network components linking multiple networks;

Figure 1C is a block diagram depicting a service provider connected to two customers that each have subnets that may contain duplicate network addresses;

Figure 2A is simplified diagram illustrating a large distributed computing enterprise environment in which the present invention is implemented;

Figure 2B is a block diagram of a preferred system management framework illustrating how the framework functionality is distributed across the gateway and its endpoints within a managed region;

Figure 2C is a block diagram of the elements that comprise the low cost framework (LCF) client component of the system management framework;

Figure 2D is a diagram depicting a logical configuration of software objects residing within a hardware network similar to that shown in **Figure 2A**;

Figure 2E is a diagram depicting the logical relationships between components within a system management framework that includes two endpoints and a gateway;

Figure 2F is a diagram depicting the logical relationships between components within a system management framework that includes a gateway supporting two DKS-enabled applications;

Figure 2G is a diagram depicting the logical relationships between components within a system management framework that includes two gateways supporting two endpoints;

Figure 3 is a block diagram depicting components within the system management framework that provide resource leasing management functionality within a distributed computing environment such as that shown in **Figures 2D-2E**;

Figure 4 is a block diagram showing data stored by a the IPOP (IP Object Persistence) service;

Figure 5A is a block diagram showing the IPOP service in more detail;

Figure 5B is a network diagram depicting a set of routers that undergo a scoping process;

Figure 6A is a block diagram showing a set of components that may be used to implement multi-customer management across multiple networks in which duplicate address may be present in accordance with a preferred embodiment of the present invention;

Figures 6B-6D are some simplified pseudo-code examples that depict an object-oriented manner in which action objects and endpoint objects can be implemented;

Figure 7A is a flowchart depicting a portion of an initialization process in which a network management system prepares for managing a set of networks with multiple NATs in accordance with a preferred embodiment of the present invention;

Figure 7B is a flowchart depicting further detail of the initialization process in which the administrator resolves addressability problems;

Figure 7C is a flowchart depicting further detail of the process in which the administrator assigns VPN IDs;

Figure 8 is a figure that depicts a graphical user interface (GUI) that may be used by a network or system administrator to set monitoring parameters for resolving address collisions in accordance with a preferred embodiment of the present invention;

Figure 9A is a flowchart showing the overall process for performing an IP "Ping" with in a multi-customer, distributed data processing system containing multiple private networks in accordance with a preferred embodiment of the present invention; and

Figure 9B is a flowchart that depicts a process by which an administrator chooses the source endpoint and target endpoint for the IP "Ping" action described in an overall manner in **Figure 9A.**

DETAILED DESCRIPTION OF THE INVENTION

The present invention provides a methodology for managing a distributed data processing system. The manner in which the system management is performed is described further below in more detail after the description of the preferred embodiment of the distributed computing environment in which the present invention operates.

With reference now to **Figure 1A**, a diagram depicts a known logical configuration of software and hardware resources. In this example, the software is organized in an object-oriented system. Application object **102**, device driver object **104**, and operating system object **106** communicate across network **108** with other objects and with hardware resources **110-114**.

In general, the objects require some type of processing, input/output, or storage capability from the hardware resources. The objects may execute on the same device to which the hardware resource is connected, or the objects may be physically dispersed throughout a distributed computing environment. The objects request access to the hardware resource in a variety of manners, e.g. operating system calls to device drivers. Hardware resources are generally available on a first-come, first-serve basis in conjunction with some type of arbitration scheme to ensure that the requests for resources are fairly handled. In some cases, priority may be given to certain requesters, but in most

implementations, all requests are eventually processed.

With reference now to **Figure 1B**, a block diagram depicts a known configuration of software and/or hardware network components linking multiple networks. A computer-type device is functioning as firewall/NAT **120**, which is usually some combination of software and hardware, to monitor data traffic from external network **122** to internal protected network **124**. Firewall **120** reads data received by network interface card (NIC) **126** and determines whether the data should be allowed to proceed onto the internal network. If so, then firewall **120** relays the data through NIC **128**. The firewall can perform similar processes for outbound data to prevent certain types of data traffic from being transmitted, such as HTTP (Hypertext Transport Protocol) Requests to certain domains.

More importantly for this context, the firewall can prevent certain types of network traffic from reaching devices that reside on the internal protected network. For example, the firewall can examine the frame types or other information of the received data packets to stop certain types of information that has been previously determined to be harmful, such as virus probes, broadcast data, pings, etc. As an additional example, entities that are outside of the internal network and lack the proper authorization may attempt to discover, through various methods, the topology of the internal network and the types of resources that are available on the internal network in order to plan electronic attacks on the network. Firewalls can prevent these types of discovery practices.

While firewalls may prevent certain entities from obtaining information from the protected internal network, firewalls may also present a barrier to the operation of legitimate, useful processes. In order to ensure a predetermined level of service, benevolent processes may need to operate on both the external network and the protected internal network. For example, a customer system is more efficiently managed if the management software can dynamically detect and dynamically configure hardware resources as they are installed, rebooted, etc. Various types of discovery processes, status polling, status gathering, etc., may be used to get information about the customer's large, dynamic, distributed processing system. This information is then used to ensure that quality-of-service guarantees to the customer are being fulfilled. However, firewalls might block these system processes, especially discovery processes.

Firewall/NAT **120** also performs network address translation between addresses on external network **122** and addresses on internal network **124**. In the example, system **130** connects to internal network **124** via NIC **132**; system **134** connects to internal network **124** via NIC **136**; and system **138** connects to internal network **124** via NIC **140**. Each NIC has its own MAC (Media Access Control layer) address, which is a guaranteed unique hardware address in the NIC that is used to address data packets to and from the system that is using a given NIC. Network Address Translator (NAT) **120** presents all of the systems on internal network **124** to external network **122** with a single, public, IP address. However, systems **130**,

134, and 138 have addresses which are unique within internal network 124. NAT 120 retrieves the addresses within the data packets flowing between the internal network and the external network, translates the addresses between the two domains, stores the addresses back into the data packets, and forwards the data packets.

The internal network supports a private address space with globally non-unique address, whereas the external network represents a public address space of globally unique addresses. A network address translator (NAT) is used to manage the connectivity of the private network with the outside world. A NAT device bridges the internal network and the external network and converts addresses between the two address spaces. Within a private network behind a NAT, an enterprise may have its own private address space without concern for integrating the private address space with the global Internet, particularly with the predominant IPv4 address space that is currently in use.

NATs are helpful for certain enterprises that do not require full connectivity for all of its devices. However, NATs present barriers for certain functionality. A NAT must have high performance in order to perform address translation on all data packets that are sent and received by a private network. In addition, a network management framework for a highly distributed system may be forced to coordinate its actions across multiple NAT devices within a single customer or across multiple customers. For example, systems 130, 134, and 138 have addresses which are unique within internal network 124.

However, another internal network within the same enterprise may duplicate the addresses that are used within internal network **124**.

When contending with multiple NATs, the network management framework cannot assume uniqueness among private network addresses. In some prior art systems, it would have been straightforward to use the private network address of a device as a unique key within the network management applications because the private network address has a unique association with a networked device. In a highly distributed system, the network management framework needs to store many data items in an efficient manner yet cannot rely upon a scheme that uses the private network addresses as unique keys for managing those data items.

Future IT solutions may not need to confront the same problems because the Internet is moving towards using a new standard IP protocol known as IP Version 6 (IPv6) that will have a much larger address space. However, a current network management solution must confront legacy issues of maintaining currently installed hardware.

Prior art solutions have generally included dedicated boxes or devices that perform address translation. These solutions tend to be specific modifications to an operating system or kernel, which reduces the benefit of having standardized implementations of software platforms. In other words, some applications may not be compatible with the solution. In addition, such solutions may require installing a dedicated device for each system, which is prohibitive.

With reference now to **Figure 1C**, a block diagram depicts a service provider connected to two customers that each have subnets that may contain duplicate network addresses. As noted above, multiple internal networks within a highly distributed data processing system may contain duplicate addresses. Service provider **150** manages networks and applications for multiple customers and stores its data within multi-customer database **152**. Customer **154** has a network of devices that includes subnet **156** that connect with the larger network through NAT **158**; customer **164** has a network of devices that includes subnet **166** that connect with the larger network through NAT **168**. Duplicate network addresses could appear within subnets **156** and **166**. In order to provide certain services in a seamless fashion such that both customers can be managed by the service provider as a single logical network, the service provider requires a network management framework that can handle duplicate network addresses.

The present invention provides a methodology for a network management framework for managing multiple networks over which duplicate addresses might appear, such as **Figure 1C**, such that the distributed network management framework is operable across multiple NATs. The manner in which the network management is performed is described further below in more detail after the description of the preferred embodiment of the distributed computing environment in which the present invention operates.

With reference now to **Figure 2A**, the present invention is preferably implemented in a large

distributed computer environment **210** comprising up to thousands of "nodes". The nodes will typically be geographically dispersed and the overall environment is "managed" in a distributed manner. Preferably, the managed environment is logically broken down into a series of loosely connected managed regions (MRs) **212**, each with its own management server **214** for managing local resources with the managed region. The network typically will include other servers (not shown) for carrying out other distributed network functions. These include name servers, security servers, file servers, thread servers, time servers and the like. Multiple servers **214** coordinate activities across the enterprise and permit remote management and operation. Each server **214** serves a number of gateway machines **216**, each of which in turn support a plurality of endpoints/terminal nodes **218**. The server **214** coordinates all activity within the managed region using a terminal node manager at server **214**.

With reference now to **Figure 2B**, each gateway machine **216** runs a server component **222** of a system management framework. The server component **222** is a multi-threaded runtime process that comprises several components: an object request broker (ORB) **221**, an authorization service **223**, object location service **225** and basic object adapter (BOA) **227**. Server component **222** also includes an object library **229**. Preferably, ORB **221** runs continuously, separate from the operating system, and it communicates with both server and client processes through separate stubs and skeletons via an interprocess communication (IPC) facility **219**. In particular, a

secure remote procedure call (RPC) is used to invoke operations on remote objects. Gateway machine **216** also includes operating system **215** and thread mechanism **217**.

The system management framework, also termed distributed kernel services (DKS), includes a client component **224** supported on each of the endpoint machines **218**. The client component **224** is a low cost, low maintenance application suite that is preferably "dataless" in the sense that system management data is not cached or stored there in a persistent manner. Implementation of the management framework in this "client-server" manner has significant advantages over the prior art, and it facilitates the connectivity of personal computers into the managed environment. It should be noted, however, that an endpoint may also have an ORB for remote object-oriented operations within the distributed environment, as explained in more detail further below.

Using an object-oriented approach, the system management framework facilitates execution of system management tasks required to manage the resources in the managed region. Such tasks are quite varied and include, without limitation, file and data distribution, network usage monitoring, user management, printer or other resource configuration management, and the like. In a preferred implementation, the object-oriented framework includes a Java runtime environment for well-known advantages, such as platform independence and standardized interfaces. Both gateways and endpoints operate portions of the system management tasks through cooperation between the client and server portions of the

distributed kernel services.

In a large enterprise, such as the system that is illustrated in **Figure 2A**, there is preferably one server per managed region with some number of gateways. For a workgroup-size installation, e.g., a local area network, a single server-class machine may be used as both a server and a gateway. References herein to a distinct server and one or more gateway(s) should thus not be taken by way of limitation as these elements may be combined into a single platform. For intermediate size installations, the managed region grows breadth-wise, with additional gateways then being used to balance the load of the endpoints.

The server is the top-level authority over all gateway and endpoints. The server maintains an endpoint list, which keeps track of every endpoint in a managed region. This list preferably contains all information necessary to uniquely identify and manage endpoints including, without limitation, such information as name, location, and machine type. The server also maintains the mapping between endpoints and gateways, and this mapping is preferably dynamic.

As noted above, there are one or more gateways per managed region. Preferably, a gateway is a fully managed node that has been configured to operate as a gateway. In certain circumstances, though, a gateway may be regarded as an endpoint. A gateway always has a network interface card (NIC), so a gateway is also always an endpoint. A gateway usually uses itself as the first seed during a discovery process. Initially, a gateway does not have any information about endpoints. As endpoints login, the gateway builds an endpoint list for

its endpoints. The gateway's duties preferably include: listening for endpoint login requests, listening for endpoint update requests, and (its main task) acting as a gateway for method invocations on endpoints.

As also discussed above, the endpoint is a machine running the system management framework client component, which is referred to herein as a management agent. The management agent has two main parts as illustrated in **Figure 2C**: daemon **226** and application runtime library **228**. Daemon **226** is responsible for endpoint login and for spawning application endpoint executables. Once an executable is spawned, daemon **226** has no further interaction with it. Each executable is linked with application runtime library **228**, which handles all further communication with the gateway.

Preferably, the server and each of the gateways is a distinct computer. For example, each computer may be a RISC System/6000™ (a reduced instruction set or so-called RISC-based workstation) running the AIX (Advanced Interactive Executive) operating system. Of course, other machines and/or operating systems may be used as well for the gateway and server machines.

Each endpoint is also a computing device. In one preferred embodiment of the invention, most of the endpoints are personal computers, e.g., desktop machines or laptops. In this architecture, the endpoints need not be high powered or complex machines or workstations. An endpoint computer preferably includes a Web browser such as Netscape Navigator or Microsoft Internet Explorer. An endpoint computer thus may be connected to a gateway via the Internet, an intranet or some other computer network.

Preferably, the client-class framework running on each endpoint is a low-maintenance, low-cost framework that is ready to do management tasks but consumes few machine resources because it is normally in an idle state. Each endpoint may be "dataless" in the sense that system management data is not stored therein before or after a particular system management task is implemented or carried out.

With reference now to **Figure 2D**, a diagram depicts a logical configuration of software objects residing within a hardware network similar to that shown in **Figure 2A**. The endpoints in **Figure 2D** are similar to the endpoints shown in **Figure 2B**. Object-oriented software, similar to the collection of objects shown in **Figure 1A**, executes on the endpoints. Endpoints **230** and **231** support application action object **232** and application object **233**, device driver objects **234-235**, and operating system objects **236-237** that communicate across a network with other objects and hardware resources.

Resources can be grouped together by an enterprise into managed regions representing meaningful groups. Overlaid on these regions are domains that divide resources into groups of resources that are managed by gateways. The gateway machines provide access to the resources and also perform routine operations on the resources, such as polling. **Figure 2D** shows that endpoints and objects can be grouped into managed regions that represent branch offices **238** and **239** of an enterprise, and certain resources are controlled by in central office **240**. Neither a branch office nor a central office is necessarily restricted to a single

physical location, but each represents some of the hardware resources of the distributed application framework, such as routers, system management servers, endpoints, gateways, and critical applications, such as corporate management Web servers. Different types of gateways can allow access to different types of resources, although a single gateway can serve as a portal to resources of different types.

With reference now to **Figure 2E**, a diagram depicts the logical relationships between components within a system management framework that includes two endpoints and a gateway. **Figure 2E** shows more detail of the relationship between components at an endpoint. Network **250** includes gateway **251** and endpoints **252** and **253**, which contain similar components, as indicated by the similar reference numerals used in the figure. An endpoint may support a set of applications **254** that use services provided by the distributed kernel services **255**, which may rely upon a set of platform-specific operating system resources **256**. Operating system resources may include TCP/IP-type resources, SNMP-type resources, and other types of resources. For example, a subset of TCP/IP-type resources may be a line printer (LPR) resource that allows an endpoint to receive print jobs from other endpoints. Applications **254** may also provide self-defined sets of resources that are accessible to other endpoints. Network device drivers **257** send and receive data through NIC hardware **258** to support communication at the endpoint.

With reference now to **Figure 2F**, a diagram depicts the logical relationships between components within a

system management framework that includes a gateway supporting two DKS-enabled applications. Gateway **260** communicates with network **262** through NIC **264**. Gateway **260** contains ORB **266** that supports DKS-enabled applications **268** and **269**. **Figure 2F** shows that a gateway can also support applications. In other words, a gateway should not be viewed as merely being a management platform but may also execute other types of applications.

With reference now to **Figure 2G**, a diagram depicts the logical relationships between components within a system management framework that includes two gateways supporting two endpoints. Gateway **270** communicates with network **272** through NIC **274**. Gateway **270** contains ORB **276** that may provide a variety of services, as is explained in more detail further below. In this particular example, **Figure 2G** shows that a gateway does not necessarily connect with individual endpoints.

Gateway **270** communicates through NIC **278** and network **279** with gateway **280** and its NIC **282**. Gateway **280** contains ORB **284** for supporting a set of services. Gateway **280** communicates through NIC **286** and network **287** to endpoint **290** through its NIC **292** and to endpoint **294** through its NIC **296**. Endpoint **290** contains ORB **298** while endpoint **294** does not contain an ORB. In this particular example, **Figure 2G** also shows that an endpoint does not necessarily contain an ORB. Hence, any use of endpoint **294** as a resource is performed solely through management processes at gateway **280**.

Figures 2F and **2G** also depict the importance of gateways in determining routes/data paths within a highly

distributed system for addressing resources within the system and for performing the actual routing of requests for resources. The importance of representing NICs as objects for an object-oriented routing system is described in more detail further below.

As noted previously, the present invention is directed to a methodology for managing a distributed computing environment. A resource is a portion of a computer system's physical units, a portion of a computer system's logical units, or a portion of the computer system's functionality that is identifiable or addressable in some manner to other physical or logical units within the system.

With reference now to **Figure 3**, a block diagram depicts components within the system management framework within a distributed computing environment such as that shown in **Figures 2D-2E**. A network contains gateway **300** and endpoints **301** and **302**. Gateway **302** runs ORB **304**. In general, an ORB can support different services that are configured and run in conjunction with an ORB. In this case, distributed kernel services (DKS) include Network Endpoint Location Service (NELS) **306**, IP Object Persistence (IPOP) service **308**, and Gateway Service **310**.

The Gateway Service processes action objects, which are explained in more detail below, and directly communicates with endpoints or agents to perform management operations. The gateway receives events from resources and passes the events to interested parties within the distributed system. The NELS works in combination with action objects and determines which gateway to use to reach a particular resource. A gateway

is determined by using the discovery service of the appropriate topology driver, and the gateway location may change due to load balancing or failure of primary gateways.

Other resource level services may include an SNMP (Simple Network Management Protocol) service that provides protocol stacks, polling service, and trap receiver and filtering functions. The SNMP Service can be used directly by certain components and applications when higher performance is required or the location independence provided by the gateways and action objects is not desired. A Metadata Service can also be provided to distribute information concerning the structure of SNMP agents.

The representation of resources within DKS allows for the dynamic management and use of those resources by applications. DKS does not impose any particular representation, but it does provide an object-oriented structure for applications to model resources. The use of object technology allows models to present a unified appearance to management applications and hide the differences among the underlying physical or logical resources. Logical and physical resources can be modeled as separate objects and related to each other using relationship attributes.

By using objects, for example, a system may implement an abstract concept of a router and then use this abstraction within a range of different router hardware. The common portions can be placed into an abstract router class while modeling the important differences in subclasses, including representing a complex system with multiple objects. With an abstracted

and encapsulated function, the management applications do not have to handle many details for each managed resource. A router usually has many critical parts, including a routing subsystem, memory buffers, control components, interfaces, and multiple layers of communication protocols. Using multiple objects has the burden of creating multiple object identifiers (OIDs) because each object instance has its own OID. However, a first order object can represent the entire resource and contain references to all of the constituent parts.

Each endpoint may support an object request broker, such as ORBs **320** and **322**, for assisting in remote object-oriented operations within the DKS environment. Endpoint **301** contains DKS-enabled application **324** that utilizes object-oriented resources found within the distributed computing environment. Endpoint **302** contains target resource provider object or application **326** that services the requests from DKS-enabled application **324**. A set of DKS services **330** and **334** support each particular endpoint.

Applications require some type of insulation from the specifics of the operations of gateways. In the DKS environment, applications create action objects that encapsulate command which are sent to gateways, and the applications wait for the return of the action object. Action objects contain all of the information necessary to run a command on a resource. The application does not need to know the specific protocol that is used to communicate with the resource. The application is unaware of the location of the resource because it issues an action object into the system, and the action object

itself locates and moves to the correct gateway. The location independence allows the NELS to balance the load between gateways independently of the applications and also allows the gateways to handle resources or endpoints that move or need to be serviced by another gateway.

The communication between a gateway and an action object is asynchronous, and the action objects provide error handling and recovery. If one gateway goes down or becomes overloaded, another gateway is located for executing the action object, and communication is established again with the application from the new gateway. Once the controlling gateway of the selected endpoint has been identified, the action object will transport itself there for further processing of the command or data contained in the action object. If it is within the same ORB, it is a direct transport. If it is within another ORB, then the transport can be accomplished with a "Moveto" command or as a parameter on a method call.

Queuing the action object on the gateway results in a controlled process for the sending and receiving of data from the IP devices. As a general rule, the queued action objects are executed in the order that they arrive at the gateway. The action object may create child action objects if the collection of endpoints contains more than a single ORB ID or gateway ID. The parent action object is responsible for coordinating the completion status of any of its children. The creation of child action objects is transparent to the calling application. A gateway processes incoming action objects, assigns a priority, and performs additional security challenges to prevent rogue action object

attacks. The action object is delivered to the gateway that must convert the information in the action object to a form suitable for the agent. The gateway manages multiple concurrent action objects targeted at one or more agents, returning the results of the operation to the calling managed object as appropriate.

In the preferred embodiment, potentially leasable target resources are Internet protocol (IP) commands, e.g. pings, and Simple Network Management Protocol (SNMP) commands that can be executed against endpoints in a managed region. Referring again to **Figures 2F and 2G**, each NIC at a gateway or an endpoint may be used to address an action object. Each NIC is represented as an object within the IPOP database, which is described in more detail further below.

The Action Object IP (AOIP) Class is a subclass of the Action Object Class. AOIP objects are the primary vehicle that establishes a connection between an application and a designated IP endpoint using a gateway or stand-alone service. In addition, the Action Object SNMP (AOSnmp) Class is also a subclass of the Action Object Class. AOSnmp objects are the primary vehicle that establishes a connection between an application and a designated SNMP endpoint via a gateway or the Gateway Service. However, the present invention is primarily concerned with IP endpoints.

The AOIP class should include the following: a constructor to initialize itself; an interface to the NELS; a mechanism by which the action object can use the ORB to transport itself to the selected gateway; a mechanism by which to communicate with the SNMP stack in a stand-alone mode; a security check verification of

access rights to endpoints; a container for either data or commands to be executed at the gateway; a mechanism by which to pass commands or classes to the appropriate gateway or endpoint for completion; and public methods to facilitate the communication between objects.

The instantiation of an AOIP object creates a logical circuit between an application and the targeted gateway or endpoint. This circuit is persistent until command completion through normal operation or until an exception is thrown. When created, the AOIP object instantiates itself as an object and initializes any internal variables required. An action object IP may be capable of running a command from inception or waiting for a future command. A program that creates an AOIP object must supply the following elements: address of endpoints; function to be performed on the endpoint, class, or object; and data arguments specific to the command to be run. A small part of the action object must contain the return end path for the object. This may identify how to communicate with the action object in case of a breakdown in normal network communications. An action object can contain either a class or object containing program information or data to be delivered eventually to an endpoint or a set of commands to be performed at the appropriate gateway. Action objects IP return back a result for each address endpoint targeted.

Using commands such as "Ping", "Trace Route", "Wake-On LAN", and "Discovery", the AOIP object performs the following services: facilitates the accumulation of metrics for the user connections; assists in the description of the topology of a connection; performs Wake-On LAN tasks using helper functions; and discovers

active agents in the network environment.

The NELS service finds a route (data path) to communicate between the application and the appropriate endpoint. The NELS service converts input to protocol, network address, and gateway location for use by action objects. The NELS service is a thin service that supplies information discovered by the IPOP service. The primary roles of the NELS service are as follows: support the requests of applications for routes; maintain the gateway and endpoint caches that keep the route information; ensure the security of the requests; and perform the requests as efficiently as possible to enhance performance.

For example, an application requires a target endpoint (target resource) to be located. The target is ultimately known within the DKS space using traditional network values, i.e. a specific network address and a specific protocol identifier. An action object is generated on behalf of an application to resolve the network location of an endpoint. The action object asks the NELS service to resolve the network address and define the route to the endpoint in that network.

One of the following is passed to the action object to specify a destination endpoint: an EndpointAddress object; a fully decoded NetworkAddress object; and a string representing the IP address of the IP endpoint. In combination with the action objects, the NELS service determines which gateway to use to reach a particular resource. The appropriate gateway is determined using the discovery service of the appropriate topology driver and may change due to load balancing or failure of primary gateways. An "EndpointAddress" object must

consist of a collection of at least one or more unique managed resource IDs. A managed resource ID decouples the protocol selection process from the application and allows the NELS service to have the flexibility to decide the best protocol to reach an endpoint. On return from the NELS service, an "AddressEndpoint" object is returned, which contains enough information to target the best place to communicate with the selected IP endpoints. It should be noted that the address may include protocol-dependent addresses as well as protocol-independent addresses, such as the virtual private network id and the IPOP Object ID. These additional addresses handle the case where duplicate addresses exist in the managed region.

When an action needs to be taken on a set of endpoints, the NELS service determines which endpoints are managed by which gateways. When the appropriate gateway is identified, a single copy of the action object is distributed to each identified gateway. The results from the endpoints are asynchronously merged back to the caller application through the appropriate gateways. Performing the actions asynchronously allows for tracking all results whether the endpoints are connected or disconnected. If the action object IP fails to execute an action object on the target gateway, NELS is consulted to identify an alternative path for the command. If an alternate path is found, the action object IP is transported to that gateway and executed. It may be assumed that the entire set of commands within one action object IP must fail before this recovery procedure is invoked.

With reference now to **Figure 4**, a block diagram

shows the manner in which data is stored by the IPOP (IP Object Persistence) service. IPOP service database **402** contains endpoint database table **404**, system database table **406**, and network database table **408**. Each table contains a set of topological (topo) objects for facilitating the leasing of resources at IP endpoints and the execution of action objects. Information within IPOP service database **402** allows applications to generate action objects for resources previously identified as IP objects through a discovery process across the distributed computing environment. **Figure 4** merely shows that the topo objects may be separated into a variety of categories that facilitate processing on the various objects. The separation of physical network categories facilitates the efficient querying and storage of these objects while maintaining the physical network relationships in order to produce a graphical user interface of the network topology.

With reference now to **Figure 5A**, a block diagram shows the IPOP service in more detail. In the preferred embodiment of the present invention, an IP driver subsystem is implemented as a collection of software components for discovering , i.e. detecting, IP "objects", i.e. IP networks, IP systems, and IP endpoints by using physical network connections. This discovered physical network is used to create topology data that is then provided through other services via topology maps accessible through a graphical user interface (GUI) or for the manipulation of other applications. The IP driver system can also monitor objects for changes in IP topology and update databases with the new topology

information. The IPOP service provides services for other applications to access the IP object database.

IP driver subsystem **500** contains a conglomeration of components, including one or more IP drivers **502**. Every IP driver manages its own "scope", which is described in more detail further below, and every IP driver is assigned to a topology manager within Topology Service **504**, which can serve may than one IP driver. Topology Service **504** stores topology information obtained from discovery controller **506**. The information stored within the Topology Service may include graphs, arcs, and the relationships between nodes determined by IP mapper **508**. Users can be provided with a GUI to navigate the topology, which can be stored within a database within the Topology Service.

IPOP service **510** provides a persistent repository **512** for discovered IP objects; persistent repository **512** contains attributes of IP objects without presentation information. Discovery controller **506** detects IP objects in Physical IP networks **514**, and monitor controller 516 monitors IP objects. A persistent repository, such as IPOP database **512**, is updated to contain information about the discovered and monitored IP objects. IP driver may use temporary IP data store component **518** and IP data cache component **520** as necessary for caching IP objects or storing IP objects in persistent repository **512**, respectively. As discovery controller **506** and monitor controller **516** perform detection and monitoring functions, events can be written to network event manager application **522** to alert network administrators of certain occurrences within the network, such as the

discovery of duplicate IP addresses or invalid network masks.

External applications/users **524** can be other users, such as network administrators at management consoles, or applications that use IP driver GUI interface **526** to configure IP driver **502**, manage/unmanage IP objects, and manipulate objects in persistent repository **512**. Configuration service **528** provides configuration information to IP driver **502**. IP driver controller **532** serves as central control of all other IP driver components.

Referring back to **Figure 2G**, a network discovery engine is a distributed collection of IP drivers that are used to ensure that operations on IP objects by gateways **260**, **270**, and **280** can scale to a large installation and provide fault-tolerant operation with dynamic start/stop or reconfiguration of each IP driver. The IPOP Service manages discovered IP objects; to do so, the IPOP Service uses a distributed database in order to efficiently service query requests by a gateway to determine routing, identity, or a variety of details about an endpoint. The IPOP Service also services queries by the Topology Service in order to display a physical network or map them to a logical network, which is a subset of a physical network that is defined programmatically or by an administrator. IPOP fault tolerance is also achieved by distribution of IPOP data and the IPOP Service among many Endpoint ORBs.

One or more IP drivers can be deployed to provide distribution of IP discovery and promote scalability of IP driver subsystem services in large networks where a

single IP driver subsystem is not sufficient to discover and monitor all IP objects. Each IP discovery driver performs discovery and monitoring on a collection of IP resources within the driver's "scope". A driver's scope, which is explained in more detail below, is simply the set of IP subnets for which the driver is responsible for discovering and monitoring. Network administrators generally partition their networks into as many scopes as needed to provide distributed discovery and satisfactory performance.

A potential risk exists if the scope of one driver overlaps the scope of another, i.e., if two drivers attempt to discover/monitor the same device. Accurately defining unique and independent scopes may require the development of a scope configuration tool to verify the uniqueness of scope definitions. Routers also pose a potential problem in that while the networks serviced by the routers will be in different scopes, a convention needs to be established to specify to which network the router "belongs", thereby limiting the router itself to the scope of a single driver.

Some ISPs may have to manage private networks whose addresses may not be unique across the installation, like 10.0.0.0 network. In order to manage private networks properly, first, the IP driver has to be installed inside the internal networks in order to be able to discover and manage the networks. Second, since the discovered IP addresses may not be unique in across an entire installation that consists of multiple regions, multiple customers, etc., a private network ID has to be assigned to the private network addresses. In the preferred embodiment, the unique name of a subnet becomes

"privateNetworkId\subnetAddress". Those customers that do not have duplicate networks address can just ignore the private network ID; the default private network ID is 0.

If Network Address Translator (NAT) is installed to translate the internal IP addresses to Internet IP addresses, users can install the IP drivers outside of NAT and manage the IP addresses inside the NAT. In this case, an IP driver will see only the translated IP addresses and discover only the IP addresses translated. If not all IP addresses inside the NAT are translated, an IP driver will not able to discover all of them. However, if IP drivers are installed this way, users do not have to configure the private network ID.

Scope configuration is important to the proper operation of the IP drivers because IP drivers assume that there are no overlaps in the drivers' scopes. Since there should be no overlaps, every IP driver has complete control over the objects within its scope. A particular IP driver does not need to know anything about the other IP drivers because there is no synchronization of information between IP drivers. The Configuration Service provides the services to allow the DKS components to store and retrieve configuration information for a variety of other services from anywhere in the networks. In particular, the scope configuration will be stored in the Configuration Services so that IP drivers and other applications can access the information.

The ranges of addresses that a driver will discover and monitor are determined by associating a subnet address with a subnet mask and associating the resulting range of addresses with a subnet priority. An IP driver is a collection of such ranges of addresses, and the

subnet priority is used to help decide the system address. A system can belong to two or more subnets, such as is commonly seen with a Gateway. The system address is the address of one of the NICs that is used to make SNMP queries. A user interface can be provided, such as an administrator console, to write scope information into the Configuration Service. System administrators do not need to provide this information at all, however, as the IP drivers can use default values.

An IP driver gets its scope configuration information from the Configuration Service, which may be stored using the following format:

```
scopeID=driverID,anchorname,subnetAddress:subnetMask[
:privateNetworkId:privateNetworkName:subnetPriority][,
subnetAddress:subnetMask:privateNetworkId:privateNetworkN
ame:subnetPriority]]
```

Typically, one IP driver manages only one scope. Hence, the "scopeID" and "driverID" would be the same. However, the configuration can provide for more than one scope managed by the same driver. "Anchorname" is the name in the name space in which the Topology Service will put the IP networks objects.

A scope does not have to include an actual subnet configured in the network. Instead, users/administrators can group subnets into a single, logical scope by applying a bigger subnet mask to the network address. For example, if a system has subnet "147.0.0.0" with mask of "255.255.0.0" and subnet "147.1.0.0" with a subnet mask of "255.255.0.0", the subnets can be grouped into a single scope by applying a mask of "255.254.0.0". Assume

that the following table is the scope of IP Driver 2. The scope configuration for IP Driver 2 from the Configuration Service would be:

2=2,ip,147.0.0.0:255.254.0.0,146.100.0.0:255.255.0.0,

69.0.0.0:255.0.0.0.

Subnet address	Subnet mask
147.0.0.0	255.255.0.0
147.1.0.0	255.255.0.0
146.100.0.0	255.255.0.0
69.0.0.0	255.0.0.0

In general, an IP system is associated with a single IP address, and the "scoping" process is a straightforward association of a driver's ID with the system's IP address.

Routers and multi-homed systems, however, complicate the discovery and monitoring process because these devices may contain interfaces that are associated with different subnets. If all subnets of routers and multi-homed systems are in the scope of the same driver, the IP driver will manage the whole system. However, if the subnets of routers and multi-homed systems are across the scopes of different drivers, a convention is needed to determine a dominant interface: the IP driver that manages the dominant interface will manage the router object so that the router is not being detected and monitored by multiple drivers; each interface is still managed by the IP driver determined by its scope; the IP address of the dominant interface will be assigned as the system address of the router or multi-homed system; and

the smallest (lowest) IP address of any interface on the router will determine which driver includes the router object within its scope.

Users can customize the configuration by using the subnet priority in the scope configuration. The subnet priority will be used to determinate the dominant interface before using the lowest IP address. If the subnet priorities are the same, the lowest IP address is then used. Since the default subnet priority would be "0", then the lowest IP address would be used by default.

With reference now to **Figure 5B**, a network diagram depicts a network with a router that undergoes a scoping process. IP driver D1 will include the router in its scope because the subnet associated with that router interface is lower than the other three subnet addresses. However, each driver will still manage those interfaces inside the router in its scope. Drivers D2 and D3 will monitor the devices within their respective subnets, but only driver D1 will store information about the router itself in the IPOP database and the Topology Service database.

If driver D1's entire subnet is removed from the router, driver D2 will become the new "owner" of the router object because the subnet address associated with driver D2 is now the lowest address on the router. Because there is no synchronization of information between the drivers, the drivers will self-correct over time as they periodically rediscover their resources. When the old driver discovers that it no longer owns the router, it deletes the router's information from the databases. When the new driver discovers the router's lowest subnet address is now within its scope, the new

driver takes ownership of the router and updates the various data bases with the router's information. If the new driver discovers the change before the old driver has deleted the object, then the router object may be briefly represented twice until the old owner deletes the original representation.

There are two kinds of associations between IP objects. One is "IP endpoint in IP system" and the other is "IP endpoint in IP network". The implementation of associations relies on the fact that an IP endpoint has the object IDs (OIDs) of the IP system and the IP network in which it is located. Based on the scopes, an IP driver can partition all IP networks, IP Systems, and IP endpoints into different scopes. A network and all its IP endpoints will always be assigned in the same scope. However, a router may be assigned to an IP Driver, but some of its interfaces are assigned to different to different IP drivers. The IP drivers that do not manage the router but manage some of its interfaces will have to create interfaces but not the router object. Since those IP drivers do not have a router object ID to assign to its managed interfaces, they will assign a unique system name instead of object ID in the IP endpoint object to provide a link to the system object in a different driver.

Because of the inter-scope association, when the IP Persistence Service (IPOP) is queried to find all the IP endpoints in system, it will have to search not only IP endpoints with the system ID but also IP endpoints with its system name. If a distributed IP Persistence Service is implemented, the IP Persistence Service has to provide extra information for searching among IP Persistence

Services.

An IP driver may use a Security Service to check the availability of the IP objects. In order to handle large number of objects, the Security Service requires the users to provide a naming hierarchy as the grouping mechanism. **Figure 5C**, described below, shows a security naming hierarchy of IP objects. An IP driver has to allow users to provide security down to the object level and to achieve high performance.

IP Driver use the concepts of "anchor" and "unique object name". An anchor is a name in the naming space which can be used to plug in IP networks. Users can define, under the anchor, scopes that belong to the same customer or to a region. The anchor is then used by the Security Service to check if a user has access to the resource under the anchor. If users want the security group define inside a network, the unique object name is used. A unique object name is in the format of:

IP network - privateNetworkID/binaryNetworkAddress

IP system - privateNetworkID/binaryIPAddress/system

IP endpoint-
privateNetworkID/binaryNetworkAddress/endppoint

For example:

A network "146.84.28.0:255.255.255.0" in privateNetworkID 12 has unique name:

12/1/0/0/1/0/0/1/0/0/1/0/1/0/1/0/0/0/0/0/1/1/1/0/0.

A system "146.84.28.22" in privateNetworkID 12 has unique name:

12/1/0/0/1/0/0/1/0/0/1/0/1/0/1/0/0/0/0/0/1/1/1/0/0/0/0/0/1/0/1/1/0/system.

An endpoint "146.84.28.22" in privateNetworkId 12 has unique name:
12/1/0/0/1/0/0/1/0/0/1/0/1/0/1/0/0/0/0/0/1/1/1/0/0/0/0/0/1/0/1/1/0/endpoint.

The IP Monitor Controller, shown in **Figure 5A**, is responsible for monitoring the changes of IP topology and objects; as such, it is a type of polling engine, which is discussed in more detail further below. An IP driver stores the last polling times of an IP system in memory but not in the IPOP database. The last polling time is used to calculate when the next polling time will be. Since the last polling times are not stored in the IPOP database, when an IP Driver initializes, it has no knowledge about when the last polling times occurred. If polling is configured to occur at a specific time, an IP driver will do polling at the next specific polling time; otherwise, an IP driver will spread out the polling in the polling interval.

The IP Monitor Controller uses SNMP polls to determine if there have been any configuration changes in an IP system. It also looks for any IP endpoints added to or deleted from an IP system. The IP Monitor Controller also monitors the statuses of IP endpoints in an IP system. In order to reduce network traffic, an IP driver will use SNMP to get the status of all IP endpoints in an IP system in one query unless an SNMP agent is not running on the IP system. Otherwise, an IP driver will use "Ping" instead of SNMP. An IP driver will use "Ping" to get the status of an IP endpoint if it is the only IP endpoint in the system since the response from "Ping" is quicker than SNMP.

With reference now to **Figure 6A**, a block diagram shows a set of components that may be used to implement multi-customer management across multiple networks in which duplicate address may be present in accordance with a preferred embodiment of the present invention. Login security subsystem **602** provides a typical authentication service, which may be used to verify the identity of users, such as administrators, during a login process. All-user database **604** provides information about all users in the DKS system, and active-user database **606** contains information about users that are currently logged into the DKS system.

Discovery engine **608**, similar to discovery controller **506** in **Figure 5**, detects IP objects within an IP network. DKS Action Object Service **610** provides action object processing within gateways. A persistent repository, such as IPOP database **612**, is updated to contain information about the discovered and monitored IP objects. Other ORB or core services **614** may also access IPOP database **612**.

Customer address manager service **616** queries IPOP **612** during operations that allow an administrator to resolve addressability problems.

Customer logical network creator **618** fetches administrator input about the groupings of physical networks into a logical network, as may be provided by an administrator through an application GUI, such as the GUI shown in **Figure 8**. From this input, the various scopes of the physical networks are combined to create a logical scope as previously described above.

VPN creator **620** fetches administrator input

concerning which physical networks belong to which customer. The administrator can provide a name to each physical network collection, which is used by the anchorname creator **622** to define an anchorname, which is highest level name used to describe a logical network. The final name of each physical network is a combination of the anchorname and the name assigned to each logical network. For example, the name of a logical network consisting of the physical networks 146.5.*.* would be "austin\downtown\secondfloor" comprising the anchorname="austin" and the name="downtown\secondfloor." The anchorname creator supplies a name to the IPDriver subsystem by combining the anchorname, determined from the scope configuration, and the name of the physical network element object from IPOP. Finally, a customer ID creator **624** uses the collection of IDs used by all customers to generate a new unique customer identifier when required by IPOP; the identifiers are used rather than strings for efficient database searches of large number of network objects. During subsequent operations to map the location of a user to an ORB, customer address manager service **616** queries active-user database **606.**

With reference now to **Figures 6B-6D**, some simplified pseudo-code depicts the manner in which action objects and endpoint objects can be implemented in an object-oriented manner. **Figure 6B** shows a class for action objects, while **Figures 6C-6D** show classes for endpoint objects.

With reference now to **Figure 7A**, a flowchart depicts a portion of an initialization process in which a network management system prepares for managing a set of networks

with multiple NATs in accordance with a preferred embodiment of the present invention. It is assumed that a network administrator has already performed configuration processes on the network such that configuration information is properly stored where necessary. The process begins when a multi-customer administrator creates DKS VPN IDs during installation (step **702**). For example, after the ORB has started, the ORB starts a Command Line Interface (CLI) Service through which an administrator can issue CLI commands to create VPN IDs with VPN DB, such as "ipop create VPN" used by customer and VPN ID creator **624** in order to create a unique customer ID or a unique VPN ID.

The process then continues with the multi-customer administrator creating network scope for one or more customers (step **704**). Multi-customer regions may also be created, which refers to the managing a region that consists of two or more customers for which care has to be taken not to intermix different customer data. At this point, the physical network is discovered via a discovery engine, such as the IP Driver Service, which performs a discovery process to identify IP objects and stored those in the IPOP persistence storage. For all customer locations, all of the physical networks that have been discovered are displayed to the administrator so that the administrator can conveniently apply names to the discovered objects/networks. In addition, multiple address problems are determined and displayed to the administrator, who is then required to assign a VPN ID to a customer, e.g., by using an application GUI such as that shown in **Figure 8**.

Part of the customer scope, i.e. a logical scope consisting of a collection of physical networks as described previously, is the customer anchorname text array, e.g., "ibm\usa\Austin", the customer name, and a unique customer ID, as created by customer address manager service **616** in **Figure 6A**. The hash number is computed by the customer base text name, e.g., "ibm", the network addresses, e.g., all subnets of reserved public addresses, and VPN IDs.

The administrator then resolves any outstanding addressability problems (step **706**). For example, a large corporation may have subnets "10.0.0.*" on each floor of a large office. After those have been resolved, then the system stores the mapping of customers, VPN IDs, customer anchornames, and customer networks in the IPOP DB (step **708**), and the initialization process is then complete.

With reference now to **Figure 7B**, a flowchart depicts further detail of the initialization process in which the administrator resolves addressability problems. **Figure 7B** provides more detail for step **706** shown in **Figure 7A** in which an administrator proceeds to resolving identified addressability problems.

The process begins, during the initialization process, as an ORB starts the customer address manager (step **712**). The customer address manager then finds the identity of the administrator that is performing various address management functions through a network management application (step **714**). At this point, the identity of the network administrator may be used to ensure than the administrator has the proper authorization parameters. However, for the sake of explanation, it may be assumed

that an administrator with multi-customer rights has access to the GUI to create VPNs for multiple customers, i.e. it may be assumed that an administrator has the proper authorization for working with the data from multiple customers. The multi-customer administrator uses the administrator GUI shown in **Figure 8**, which uses the customer address manager, to display all of the discovered networks for the administrator's customer or customers (step **716**).

After retrieving this information, the customer address manager may then allow the administrator to assign VPN IDs to those networks for which it can be determined that the networks have an addressability problem (step **718**). The assigned VPN IDs are then stored as updated information within the network objects within IPOP (step **720**). The scope information is also updated with a VPN ID (step **722**); initially, many scopes are defined as "VPN = 0", which means no VPN address. The VPN ID creator ensures that unique VPN IDs are created such that duplicate addresses can exist within a VPN that has an assigned VPN ID. This portion of the initialization process is then complete. The manner in which the administrator assigns VPN IDs is explained in more detail with respect to **Figure 7C**.

In order to determine which networks require a VPN ID, the customer address manager sorts through all of the network addresses, looking for problematic addresses. For example, a set of 255 public addresses, such as "10.0.0.*", are reserved for local network purposes. Hence, even if two networks within the network management system do not have colliding local network addresses, the

potential for future collisions exists.

With reference now to **Figure 7C**, a flowchart depicts further detail of the process in which the administrator assigns VPN IDs. **Figure 7C** provides more detail for step **718** shown in **Figure 7B**. The process begins by displaying those networks have been determined to need a VPN ID assigned since a duplicate address exists, as determined with respect to step **718** above, to the current administrator (step **732**). The customer address manager then displays a list of possible VPN IDs from which the administrator may choose (step **734**), and the administrator is able to define VPN IDs as necessary if not already defined (step **736**). VPN IDs could have been previously defined through the configuration service, most likely during installation. However, at configuration time, the networks have not yet been discovered, so it is not possible for the system to know if and where duplicate addresses exist. While the figures are described with respect to the actions of a single administrator, a highly distributed system has a collection of administrators that are typically not in one location. Hence, one of goals of the DKS management framework is to detect errors and allow the administrators to have input into the manner in which the errors should be corrected.

A determination is then made as to whether the administrator is a multi-customer administrator (step **738**). If not, then the VPN ID that has been chosen by the administrator can be assigned to the networks of the administrator's customer (step **740**). If the administrator is a multi-customer administrator, then the

customer address manager must get a specific customer from the administrator (step **742**), and the chosen VPN ID is assigned to the specified customer (step **744**). This portion of the initialization process is then complete. After these initialization steps, the administrator has an overall addressing scheme that should be coherent. The IP addresses, VPN IDs, and other information, when taken together, provides a scheme for unique identifiers that the management system can use to manage the devices throughout the system.

With reference now to **Figure 8**, a figure depicts a graphical user interface window that may be used by a network or system administrator to set monitoring parameters for resolving address collisions in accordance with a preferred embodiment of the present invention. Window **850** is a dialog box that is associated with a network management application; a system or network administrator is required to create or enter VPN IDs to resolve duplicate addresses that have been detected, such as physical network addresses **852** and **854**. An administrator could also invoke the application on a regular basis when necessary, or it could be invoked automatically by the network management system when address collisions are detected. "Set" button **874** and "Clear" button **876** allow the administrator to store the specified values or to clear the specified parameters. Checkbox **878** allows an administrator to quickly change the VPN ID for an entire physical scope indicated within window **850**.

Figures 9A-9B depict examples of processes that may be performed by the network management system after

system configuration/initialization when an administrator is using a network management application to perform a certain operation, such as a simple IP "Ping" command as shown in the example. While the example shows a simple IP "Ping" action, a more complex action could include a software distribution application that installs software on endpoints throughout the distributed data processing system.

With reference now to **Figure 9A**, a flowchart shows the overall process for performing an IP "Ping" within a multi-customer, distributed data processing system containing multiple private networks in accordance with a preferred embodiment of the present invention. The process begins when an ORB starts a private network multi-customer manager (PNMCM) that is used by the system to perform certain actions, such as requesting an IP "Ping" (step **902**). The user of the application, which in this case is a network or system administrator for a particular customer, launches an application associated with the PNMCM (step **904**). Within the application, the administrator chooses an endpoint and requests an IP "Ping" action, most likely from hitting a "Ping" button within the GUI (step **906**).

The PNMCM manager attempts to fetch the requested endpoint from the IPOP database using only the IP address as specified or selected by an administrator within an application GUI (step **908**). A determination is then made as to whether IPOP returns duplicate endpoints (step **910**). If not, then the process branches to show the results of the requested "Ping" action.

If there is a collision among duplicate IP

addresses, they are displayed to the administrator along with the previously associated VPN IDs that help to uniquely identify the endpoints (step **912**). The administrator is requested to choose only one of the duplicate endpoints (step **914**), and after choosing one, the administrator may request to perform the "Ping" action on the selected endpoint (step **916**). The PNMCM displays the results of the "Ping" action to the administrator (step **918**), and the process is complete.

With respect to **Figure 9B**, a flowchart depicts a process for obtaining and using an application action object (AAO) within the network management system of the present invention. An application action object is a class of objects that extends an action object class in a manner that is appropriate for a particular application.

The process begins when an application requests, from the Gateway service, an application action object (AAO) for a "Ping" action (AAOIP) against a target endpoint (step **922**). The process assumes that the administrator has already chosen the source and target endpoints through some type GUI within a network management application.

The gateway service asks the NEL service to decode the target endpoint from the request (step **924**). As noted previously, one of the primary roles of the NEL service is to support the requests from applications for routes, as explained above with respect to **Figure 3**. The NEL service then asks the IPOP service to decode the endpoint object (step **926**). Assuming that the processing has been successfully accomplished, IPOP returns an appropriate AAOIP object to the NEL service, including

VPN ID if necessary (step **928**), and the NEL service returns the AAOIP object to the Gateway service (step **930**). The Gateway service then returns the AAOIP object to the application (step **932**). The application then performs the desired action (step **934**), such as an IP "Ping", and the process is complete.

The advantages of the present invention should be apparent in view of the detailed description of the invention that is provided above. The present invention provides a flexible network management framework in a highly distributed system with significant potential for duplicate addresses such that the network management framework can handle the intermingling of addresses from multiple customer networks and provide the ability to manage multiple customers within a single logical network.

Automatic correction of duplicate addresses is important so that two NIC cards do not receive data intended only for one of them. Duplicate network addresses may continue to exist within the system while the network management system maintains unique system names or identifiers for the endpoints; the system maps addresses in the form of network addresses and VPN IDs to user friendly names but does not depend on them for uniqueness. The system manages devices without using dedicated hardware devices throughout the system, such as specially configured switches, gateways, and hubs.

The network management framework allows logical networks to be determined within the physical networks of the distributed system. The framework does not require operating system kernel changes and prevents unintended

routing by other operating system layers. The management components are transparent since object IDs, such as an IPOPOid, is used as an application action object; the applications do not need to know how to decode objects and addresses, which the gateway service and IPOP service do on behalf of the applications.

It is important to note that while the present invention has been described in the context of a fully functioning data processing system, those of ordinary skill in the art will appreciate that the processes of the present invention are capable of being distributed in the form of instructions in a computer readable medium and a variety of other forms, regardless of the particular type of signal bearing media actually used to carry out the distribution. Examples of computer readable media include media such as EPROM, ROM, tape, paper, floppy disc, hard disk drive, RAM, and CD-ROMs and transmission-type media, such as digital and analog communications links.

The description of the present invention has been presented for purposes of illustration but is not intended to be exhaustive or limited to the disclosed embodiments. Many modifications and variations will be apparent to those of ordinary skill in the art. The embodiments were chosen to explain the principles of the invention and its practical applications and to enable others of ordinary skill in the art to understand the invention in order to implement various embodiments with various modifications as might be suited to other contemplated uses.